## Using KNN to manually calculate the distance

This is the training data and the test data:

| Accelerometer Data | | | Gyroscope Data | | | Fall (+), Not (-) |
|---|---|---|---|---|---|---|
| x | y | z | x | y | z | +/- |
| 1 | 2 | 3 | 2 | 1 | 3 | - |
| 2 | 1 | 3 | 3 | 1 | 2 | - |
| 1 | 1 | 2 | 3 | 2 | 2 | - |
| 2 | 2 | 3 | 3 | 2 | 1 | - |
| 6 | 5 | 7 | 5 | 6 | 7 | + |
| 5 | 6 | 6 | 6 | 5 | 7 | + |
| 5 | 6 | 7 | 5 | 7 | 6 | + |
| 7 | 6 | 7 | 6 | 5 | 6 | + |
| 7 | 6 | 5 | 5 | 6 | 7 | ?? |

Answer:

K=sqrt(samples number)= sqrt(8)=2.8284=3

How to calculate the distance: (Target x1- Data x1)^2 + (Target x2- Data x2)^2

(7-1)^2+(6-2)^2+(5-3)^2+(5-2)^2+(6-1)^2+(7-3)^2=106

(7-2)^2+(6-1)^2+(5-3)^2+(5-3)^2+(6-1)^2+(7-2)^2=108

(7-1)^2+(6-1)^2+(5-2)^2+(5-3)^2+(6-2)^2+(7-2)^2=115

(7-2)^2+(6-2)^2+(5-3)^2+(5-3)^2+(6-2)^2+(7-1)^2=101

(7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2=6

(7-5)^2+(6-6)^2+(5-6)^2+(5-6)^2+(6-5)^2+(7-7)^2=7

(7-5)^2+(6-6)^2+(5-7)^2+(5-5)^2+(6-7)^2+(7-6)^2=10

(7-7)^2+(6-6)^2+(5-7)^2+(5-6)^2+(6-5)^2+(7-6)^2=7

Choose smallest 3 number:

(7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2=6 +

(7-5)^2+(6-6)^2+(5-6)^2+(5-6)^2+(6-5)^2+(7-7)^2=7 +

(7-7)^2+(6-6)^2+(5-7)^2+(5-6)^2+(6-5)^2+(7-6)^2=7 +

The final answer will be

7 6 5 5 6 7 +

## Knn with python:

knn.ipynb

文件 修改 视图 插入 代码执行程序 工具 帮助 已保存所有更改

+ 代码 + 文本

```python
# Example of making predictions
from math import sqrt

# calculate the Euclidean distance between two vectors
#     Euclidean Distance = sqrt(sum i to N (x1_i – x2_i)^2)
# use sqrt
def euclidean_distance(row1, row2):
  distance = 0.0
  for i in range(len(row1)-1):
    distance += (row1[i] - row2[i])**2
  return sqrt(distance)


def get_neighbors(train, test_row, num_neighbors):
  distances = list()
  for train_row in train:
    dist = euclidean_distance(test_row, train_row)
    distances.append((train_row, dist))
  distances.sort(key=lambda tup: tup[1])
  neighbors = list()
  for i in range(num_neighbors):
    neighbors.append(distances[i][0])
  return neighbors

  # Make a classification prediction with neighbors
# - test_row is row 0
# - num_neighbors is 3
def predict_classification(train, test_row, num_neighbors):
  neighbors = get_neighbors(train, test_row, num_neighbors)
  output_values = [row[-1] for row in neighbors]
  prediction = max(set(output_values), key=output_values.count)
  return prediction

# Test distance function, 0 means not fall(-) and 1 means fall(+)
dataset = [[1, 2, 3, 2, 1, 3, 0],
  [2, 1, 3, 3, 1, 2, 0],
  [1, 1, 2, 3, 2, 2, 0],
  [2, 2, 3, 3, 2, 1, 0],
  [6, 5, 7, 5, 6, 7, 1],
  [5, 6, 6, 6, 5, 7, 1],
  [5, 6, 7, 5, 7, 6, 1],
  [7, 6, 7, 6, 5, 6, 1]]
```

```python
# Test distance function, 0 means not fall(-) and 1 means fall(+)
dataset = [[1, 2, 3, 2, 1, 3, 0],
  [2, 1, 3, 3, 1, 2, 0],
  [1, 1, 2, 3, 2, 2, 0],
  [2, 2, 3, 3, 2, 1, 0],
  [6, 5, 7, 5, 6, 7, 1],
  [5, 6, 6, 6, 5, 7, 1],
  [5, 6, 7, 5, 7, 6, 1],
  [7, 6, 7, 6, 5, 6, 1]]

# enter the testdata and i predict it will be fall(+)
testdata = [[7, 6, 5, 5, 6, 7, 0]]

prediction = predict_classification(dataset,testdata[0], 3)

# - dataset[0][-1] is the last element of row 0 of dataset
# - Display
#    Expected 0, Got 0.
print('Expected %d, Got %d.' % (dataset[0][-1], prediction))
```

```
Expected 0, Got 1.
```

Conclusion:

It will be easier to calculate by hand when the amount of the testing data is small. For huge data, python will be faster and more accurate.